REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. REPORT DATE *(DD-MM-YYYY)*	2. REPORT TYPE	3. DATES COVERED (From – To)
26-11-2001	FINAL	26-11-2001 – 26-11-2001

4. TITLE AND SUBTITLE

Software Change and Regression Testing

5a. CONTRACT NUMBER
n/a

5b. GRANT NUMBER
n/a

5c. PROGRAM ELEMENT NUMBER
n/a

6. AUTHOR(S)

LT Alex Hoover, USN

5d. PROJECT NUMBER
n/a

5e. TASK NUMBER
n/a

5f. WORK UNIT NUMBER
n/a

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)

Commander, Operational Test and Evaluation Force
7970 Diven Street
Norfolk, VA 23505

8. PERFORMING ORGANIZATOIN REPORT NUMBER
n/a

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)

Commander, Operational Test and Evaluation Force
7970 Diven Street
Norfolk, VA 23505

10. SPONSOR/MONITOR'S ACRONYM(S)
COMOPTEVFOR

11. SPONSOR/MONITOR'S REPORT NUMBER(S)
n/a

12. DISTRIBUTION/AVAILABILITY STATEMENT

Unlimited

13. SUPPLEMENTARY NOTES

none

14. ABSTRACT

The only constant is change. Nowhere is this more evident than in the world of software.

The addition of field changes, patches, updates and upgrades at regular, short-term intervals makes the acquisition of software intensive systems complex at best. Is the system configuration as stable today in test as it was six months ago at the last software review? Is today's test item representative of the item that will be delivered to the servicemember in the field a year from now?

Traditional software metrics fall short of providing useful answers to these questions. SLOC, version numbers, and release dates describe the physical characteristics of an application. The questions that most often arise in the management of a software program, however, need information on the operational performance of the application.

15. SUBJECT TERMS

software, metrics, regression, operational test

16. SECURITY CLASSIFCATION:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON
UNCLASSIFIED				6	
b. REPORT	b. ABSTRACT	c. THIS PAGE			19b. TELEPHONE NUMBER (include area code)

Standard Form 298 (Rev. 8-98)
Prescribed by ANSI Sta. 239-18
298-102

GENERAL INSTRUCTIONS FOR COMPLETING SF 298

The Report Documentation Page (RDP) is used in announcing and cataloging reports. It is important that this information be consistent with the rest of the report, particularly the cover and title page. Instructions for filling in each block of the form follow. It is important to ***stay within the lines*** to meet ***optical scanning requirements.***

Block 1. Agency Use Only *(Leave blank).*

Block 2. Report Date. Full publication date including day, month, and year, if available.
(e.g. 1 Jan 88). Must cite at least the year.

Block 3. Type of Report and Dates Covered. State whether report is interim, final. Etc. If applicable, enter inclusive report dates (e.g. 10 June 87 – 30 Jun 88).

Block 4. Title and Subtitle. A title is taken from the part of the report that provides the most meaningful and complete information. Wehn a report is prepared in more than one volume, repeat the primary title, add volume number, and include subtitle for the specific volume. On classified documents enter the title classification in parentheses.

Block 5. Funding Numbers. To include contract and grant numbers; may include program element number(s), project number(s), task number(s), and work unit number(s). Use the following labels:

C – Contract	**PR** – Project
G – Grant	**TA** – Task
PE – Program Element	**WU** – Work Unit Accession No.

Block 6. Author(s). Name(s) of person(s) responsible for writing the report, performing the research, or credited with the content of the report. If editor or compiler, this should follow name(s).

Block 7. Performing Organization Name(s) and Address(es). Self-explanatory.

Block 8. Performing Organization Report Number. Enter the unique alphanumeric report number(s) assigned by the organization performing the report.

Block 9. Sponsoring/Monitoring Agency Name(s) and Address(es). Self-explanatory.

Block 10. Sponsoring/Monitoring Agency Report Number. *(If known)*

Block 11. Supplementary Notes. Enter information not included elsewhere such as: Prepared in cooperation with...; Trans. Of ...; To be published in ... When a report is revised, include a statement whether the new report supersedes or supplements the older report.

Block 12a. Distribution/Availability Statement. Denotes public availability or limitations. Cite any availability to the public. Enter additional limitations or special markings in all capitals (e.g. NOFORN, REL, ITAR).

DOD – See DoDD 5230.24, "Distribution Statements on Technical Documents."
DOE – See authorities.
NASA – See Handbook NHB 2200.2.
NTIS – Leave blank.

Block 12b. Distribution Code.

DOD – Leave blank.
DOE – Enter DOE distribution categories from the Standard Distribution for Unclassified Scientific and Technical Reports..
NASA – Leave blank.
NTIS – Leave blank.

Block 13. Abstract. Include a brief *(Maximum 200 words* factual summary of the most significant information contained in the report.

Block 14. Subject Terms. Keywords or phrases identifying major subjects in the report.

Block 15. Number of Pages. Enter the total number of pages.

Block 16. Price Code. Enter appropriate price code *(NTIS only).*

Block 17. – 19. Security Classifications. Self-explanatory. Enter U.S. Security Classification in accordance with U.S. Security Regulations (i.e. UNCLASSIFIED). If form contains classified information, stamp classification on the top and bottom of the page.

Block 20. Limitation of Abstract. This block must be completed to assign a limitation to the abstract. Enter either UL (unlimited) or SAR (same as report). An entry is this block is necessary if the abstract is to be limited. If blank, the abstract is assumed to be unlimited.

Software Change and Regression Testing

LT Alex Hoover, LT USN
Undersea Warfare Division
Commander, Operational Test and Evaluation Force
7970 Diven Street
Norfolk, VA 23505
(757) 444-5546 x3397
hoovera@cotf.navy.mil

69th MORS Symposium
Working/Composite Group number (25 B) or Special Session
12-14 June 2001
(26 November 2001)

Background

The only constant is change. Nowhere is this more evident than in the world of software.

The addition of field changes, patches, updates and upgrades at regular, short-term intervals makes the acquisition of software intensive systems complex at best. Is the system configuration as stable today in test as it was six months ago at the last software review? Is today's test item representative of the item that will be delivered to the servicemember in the field a year from now?

Traditional software metrics fall short of providing useful answers to these questions. SLOC, version numbers, and release dates describe the physical characteristics of an application. The questions that most often arise in the management of a software program, however, need information on the operational performance of the application. To answer this type of question in a meaningful way, we need to make two evaluations: (1) We must assess the performance based impact of a change, and (2) We must determine what level of testing is required for the assessed performance impact.

This paper has three main points: (1) describe a framework and a set of terms of reference for discussing the operational impact of change in a software application, (2) Provide an example of applying this framework to develop change metrics for a specific problem, and (3) illustrate the evaluation of these metrics to make a programmatic decision.

Study Case

Problem Statement

The specific problem under discussion will require an evaluation of the performance based impact of a code port from Ada to C++ (an increasingly common event) for the acoustic processors of a torpedo control system and investigate the nature of understanding and developing such

impact based evaluations. The criteria for use in this evaluation are Total Architectural Change (TAC), a top-level metric that describes the degree of impact the code port will have on torpedo performance, and Code Change (CC), a working-level metric that describes the extent of change within the components. TAC and CC will be used to determine the amount and type of regression testing required as a result of the code port.

Assumptions

Only a code port is being tested. There are no changes in the host platform, operating system, peripheral interfaces, torpedo body, or warhead characteristics. If this assumption is not true, the tests prescribed by this report will not comprehensively address required regression testing of the torpedo.

The Ada code is implemented using object oriented (OO) methodology (Ada 95 or later). If this assumption is not true, the code port should be treated as if the Total Architectural Change (TAC) is 100%.

Testing Requirements

Risk Areas

Architecture

Various design decisions made in the development of the software (component communications, queuing, object hierarchy, data structures, etc.) for the Ada design may or may not be appropriate for use with a C++ implementation. These attributes of the architecture must be analyzed with respect to latency, data integrity, modifiability, availability, reliability, and stability to determine to what degree the design must be changed in order to meet the functional requirements of the ADCAP. The degree of change in the design will have a direct impact on the changes required in the code.

TAC is calculated as the percentage of Critical Software Components (CSC's) that undergo significant change as a result of the code revision. The two main tasks in calculating TAC, then, are to determine at what resolution to distinguish CSC's and to identify what is a significant change.

Division of CSC's

The TAC is to have a macroscopic effect on the testing requirements, determining whether full-scale testing of the system will be required or the system can meet regression criteria through gradated testing based on code change. To achieve this behavior, the TAC should vary linearly with respect to change in software function. Programmatically, applications are often divided into functional code segments that are designed by a single team from a core set of algorithms. These segments often have one-to-one interface relationships which are stable parts of the architecture. These properties normally lead to the linear behavior desired for TAC, and thus identify program functional code segments are good candidates for CSC's.

To formally validate linearity, investigate the next higher and lower possible sets of code divisions. Linear behavior will be exhibited when changing the implementation of a code segment at the next lower level

necessitates changing the implementation of peer code segments. At the highest level at which linearity is achieved, a change in one code segment does not necessitate a change in peer code segments, making TAC evaluated at the next higher level of division different from the candidate level.

In application, most types of code never exhibit strict linearity in this way, so an engineering judgement call is required to determine the highest level at which TAC remains stable when different segments are changed.

Significant Change

There are various differences between the Ada and C++ languages that will necessitate functional differences in the implementation of the system. The DoD Requirements for High Order Computer Programming Languages (DOD 1976) should be used to gauge functional changes in the code. If the Ada code exploits a Steelman requirement not provided in C++, the code should be considered significantly changed. Note that the Steelman is not being used to determine the suitability of either language for the intended purpose, but rather to provide a common basis of comparison for the functional aspects of different implementations of the same design in the two languages.

Code Change

Code Change should be accounted at the lowest level possible for the application, in this case along the domain of object methods and properties. A method or property that requires one or more significant changes, as described above, is considered changed. The measure of object change, δ_o, is the number of changed methods and properties divided by the total number of methods and properties.

$$\delta_o = \frac{\text{changed methods + changed properties}}{\text{total methods + total properties}}$$

The first-degree extent of change, ext^1, is the number of object interfaces that change involves divided by the total number of object interfaces. An object interface is defined as the relationship between two objects where one object activates the methods or queries the properties of another.

$$ext^1 = \frac{\text{number of changed object interfaces}}{\text{total number of object interfaces}}$$

The Code Change within the software system, Δ, used to evaluate risk for this code port is the product of the total object change and the first-degree extent of change:

$$\Delta = \delta_o \times ext^1$$

The total code change will determine gradation of testing requirements as appropriate for the determined level of change.

Types of Test

The purpose of determining the TAC and CC for an application undergoing revision is to direct the test efforts for that system to achieve maximum confidence in system performance while incurring minimum cost. Listed below is a representative subset of different testing techniques (DMSO 2001) that can be used to verify the regression performance of the torpedo controller after the revision to the acoustic

functional segment. Each type of test requires different resources and looks at performance in different ways.

Functional Component Test

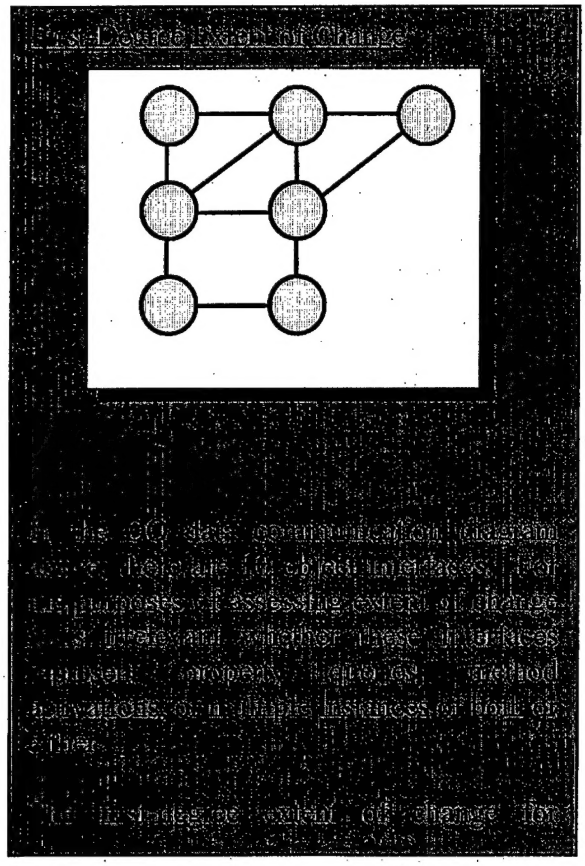
A full Functional Component Test requires testing the performance of each CSC against its requirements. A partial Functional Component Test requires testing the performance of CSCs in which an object has undergone significant change.

Integration Test

An Integration Test tests the interaction among CSCs, ensuring functional correctness and the ability to meet quality requirements (latency, stability, reliability, etc.) of the design. Thread testing is exemplary of an integration test. Functional component tests along with integration tests provide the basis for acceptance of the modified system for operational test.

Turing Test

In a Turing test, two systems are run under identical conditions. Subject matter experts compare the performances of the systems in an attempt to identify any distinctions between the systems. For the purposes of a Turing test on the torpedo code port, the Naval Undersea Warfare Center, Newport RI Weapons Assessment Facility modeling and simulation application could be an acceptable test environment. The test plan for such a test would specify the required number of tests, initial conditions of the test scenarios, performance metrics to be compared, and the subjective and objective criteria



for comparison. While the Turing test is required to produce a high correlation between the systems' performances, the number of tests conducted determines the confidence of the result.

The Turing test can only be used to ensure consistency of performance and not to identify decline or improvement in performance. If the code port results in a significant improvement in performance, the system will fail the Turing test. Failure of the Turing test with improved performance in the simulated environment does not equate to confidence in improved performance in the real world environment. Any failure of the Turing test would require in water testing to resolve the regression requirement for the torpedo.

Change		Functional Component Test	Integration Test	Turing Test	In Water Test
TAC > .2		Full	Mandatory	N/A	Mandatory
TAC ≥ .2	.8 < Δ	Full	Mandatory	N/A	Mandatory
	.6 < Δ < .8	Full	Mandatory	High Confidence	Dependent
	.4 < Δ < .6	Full	Mandatory	Low Confidence	Dependent

In Water Test

The In Water Test is standard COMOPTEVFOR at sea testing. If the changes in the system due to the code port are so large or if the results of less robust testing are confounded, as described above, in water testing will be required to resolve the regression performance issue.

Determining the Level of Test

By accepting a test method that takes a more focused look at the system and uses fewer resources (though not necessarily at lower organizational cost), the test authority accepts greater risk of ambiguous results that would require further testing.

TAC and CC can be used to provide a high confidence estimate of the minimum testing requirements for a given code revision. Total Architectural Change is used a first level index of the operational impact of the code revision. A nominal threshold value must be determined, above which a full, in-water test is required. This threshold roughly equates to the percentage of variability that the end user of the application is willing to accept between the old system and the new system regression performance. Thus, if the torpedo in question had a 50% probability of hit and the customer were willing to accept ± 5% variability, the TAC full test threshold would be .2 (that is the range of variability, 10% is .2 of the 50% performance characteristic).

If the application is determined to be a candidate for graduated testing, the degree of testing should be determined by the degree of Code Change. A greater CC value indicates a greater degree of change with respect to the operational characteristics of the application. Thus, the degree and type of testing required to validate the new software against regression performance increases with increasing CC. A representative table showing the degree of testing required is given above.

Summary

While change to software systems, on the whole, is difficult to describe, it is possible to quantify the impact of change to software on different programmatic activities. With respect to regression performance testing, the change to a software system due to code revision should be evaluated at the macroscopic and working levels.

The macroscopic evaluation of software change is based upon significant change to high level Critical Software Components. At this level the impact of

change is proportional to the changes in the system design.

The working level evaluation of change is based upon both the volume of change effected in the code and the degree of interdependence within the code. Change to a few lines of code that are heavily accessed by the rest of the application is weighted as much as extensive change to relatively independent pieces of code.

Both the macroscopic and the working level degrees of impact need to be considered when determining the extent of testing required to validate regression performance of the application.

References

Department of Defense. *The DoD Requirements for High Order Computer Programming Languages (Steelman).* 1976.

Defense Modeling and Simulation Office. *DMSO Recommended Practices Guide.* 2000.

Bibliography

Stevens, Roger T. *Operational Test and Evaluation: A Systems Engineering Process.* Krieger Publishing Company. Malabar FL. 1986.

DeWitt, R. N. *Principles of Testing a Data Fusion System.* Pacific-Sierra Research, Inc. 1998.

Biography

Lieutenant Alex Hoover is an Operational Test Coordinator and Modeling and Simulation Analyst in the Undersea Warfare division of Commander, Operational Test and Evaluation Force. He has six years of at sea experience working with software systems in support of the Navy's Combat Logistics and Cruiser/Destroyer fleets. He holds a BS CIS from the department of Engineering at the Ohio State University and an MS CIS from the University of Phoenix.